第 1 章

Rust 语言简介

小张在一次嵌入式应用开发工作中，遇到了令他无比苦恼的问题：在应用功能正常的情况下，运行一段时间后总会出现系统进程卡死的情况。小张进行了多次代码审查，但始终找不到问题的根源。于是，感到摸不着头脑的小张只好去请教自己的师傅大周。

大周根据小张的描述，重点查看了进程的内存资源使用情况，发现内存占用异常增高。每当内存占满时，应用进程就会卡死。大周对小张说：“应该是发生了内存泄漏，重点检查一下内存块是否被正确释放。”小张通过代码审查发现，确实存在这个问题。

```
// 定义获取芯片校验和的函数，参数为起始地址和大小
DWORD ChipHandler::Getchipchecksum(DWORD dwstart, int nsize)
{
    DWORD dwchecksum = 0;                // 初始化校验和为 0
    BYTE *pData = (BYTE*)malloc(nsize); // 分配内存以存储读取的数据
    // 从指定地址读取内存，如果失败
    if(ReadMem(dwstart, pData, nsize) == false)
    {
        free(pData);                     // 释放之前分配的内存
        LOGERROR("Get Checksum error"); // 记录错误信息
        return 0;                        // 返回 0 表示出现错误
    }
    for(int i = 0; i < nsize; i++)       // 遍历读取的数据
    {
        dwchecksum += pData[i];          // 累加每个字节到校验和
    }
    return dwchecksum;                   // 返回计算得到的校验和
}
```

问题虽然被解决了，但如何预防类似情况的发生呢？

针对这个问题，技术总监 Tom 给小张布置了一个任务：找到一种方法来预防代码中的内存泄漏。同时，他给小张介绍了 cppcheck、valgrind、splint 等代码扫描工具，并要求他尝试这

些工具是否能帮助找出问题。

小张一一尝试了这些工具，结果发现静态检测未能发现代码中的问题，甚至动态检测还存在代码覆盖率不足的问题，无法完美解决问题。

经过多轮讨论，团队决定：

（1）加强代码的规范管理：内存块的申请和释放统一在模块的头/尾部处理。

（2）加强代码审查：对于每个新增的模块，进行 1:1 的检查。

（3）增加针对内存使用情况的测试用例。

Tom: “这些措施都是亡羊补牢的做法。我们目前身处汽车电子行业，需要对车辆和乘客的安全负责，必须找到根本的解决方法。”

“也许我们可以尝试一下 Rust。”大周说道。

“Rust？几年前我看过一些文章介绍这个语言，据说它要替代 C/C++的现代语言，优势是内存安全。看起来应该能解决我们的问题，但不知道它是否适合汽车电子行业，是否适合我们团队。”Tom 心里想道，接着布置了下一个任务：

“听起来不错，周工带着小张成立一个小组，专题研究一下 Rust，目的是检验 Rust 是否适合在汽车电子行业替代 C/C++。”

就这样，老周这个多年的嵌入式开发老手带着小张这个嵌入式开发新人，开始了 Rust 的研究之旅。

1.1　Rust 语言概述

Rust 最早是由 Mozilla Research 中的 Graydon Hoare 个人开发的一种系统编程语言。在 Graydon Hoare 的创意和努力下，Rust 逐渐成形，并最终得到了 Mozilla Research 的认可和支持。Rust 首次发布于 2010 年，在其后的几年里，随着不断地发展和完善，Rust 逐渐在编程界崭露头角。

Rust 是一种系统级编程语言，具有内存安全、并发性和高性能的特点。它的设计哲学是结合安全性而不牺牲性能，旨在替代 C 和 C++，解决这些语言长期存在的内存安全问题。自首次亮相以来，Rust 迅速赢得了许多开发者的青睐，越来越多的企业和项目开始采用 Rust 作为主要编程语言，涵盖了操作系统、数据库、游戏、区块链、云计算等多个领域。

此外，Rust 语言还有官方构建系统和包管理器 Cargo。类似于其他编程语言中的 npm（Node.js）、Maven（Java）或 pip（Python）。Cargo 是一个用于管理 Rust 项目的命令行工具，提供了一系列功能，以简化项目的构建、依赖管理和发布过程。

Cargo 的一些主要特点包括：

- 依赖管理：Cargo 允许开发者在项目中添加依赖项，并自动下载和管理这些依赖项的版本。

- 构建系统：Cargo 提供了一个简单易用的构建系统，可以编译、运行和测试 Rust 代码，还可以生成文档。
- 项目管理：Cargo 通过自动生成标准项目结构和简化配置文件，帮助开发者创建、组织和管理 Rust 项目。
- 包发布：Cargo 可以将项目发布到 crates.io（Rust 的官方包管理仓库），使其可供其他开发者使用。

使用 Cargo，开发者可以更高效地管理项目的依赖、构建和发布过程。同时，Cargo 也是开发者非常重要且核心的工具，可以帮助他们更快速地开发和维护 Rust 项目。Cargo 的设计简单、易用，促进了 Rust 社区的库和工具生态系统的繁荣和健壮。

总的来说，Rust 在编程界逐渐崭露头角，吸引了越来越多的开发者和企业的关注和青睐。可以预见，这门编程语言在未来有着更广阔的发展空间，并将为编程领域带来新的可能性。

1.2　Rust 语言特性

1. 内存安全性

提到 Rust 语言的特性，首先离不开的便是内存安全性。

Rust 的内存安全性主要体现在所有权（ownership）系统、借用（borrowing）检查器和生命周期（lifetimes）上。这三个概念相互配合，确保程序在运行时不会出现内存泄漏、数据竞争或空指针引用等问题。

所有权系统是 Rust 的核心特性之一。在 Rust 中，规定每个数据只能有一个所有者。当值被绑定到一个变量时，该变量即成为值的所有者。当所有者超出作用域时，Rust 会自动调用析构函数来释放内存。这种所有权机制避免了内存泄漏的问题，因为在运行时会自动释放不再需要的内存。所有权规则还允许 Rust 在不需要垃圾收集器的情况下有效管理内存。

借用检查器是指 Rust 在实施所有权规则的基础上，允许数据的借用。这意味着可以在不转移所有权的情况下访问值，而无须创建数据的副本。借用检查器在编译时检查代码中的不可变借用和可变借用。不可变借用允许多个读者同时访问数据，而可变借用则确保在某个作用域内只有一个线程可以修改数据。这保证了在同一时间内只有一个可变引用或多个不可变引用，从而避免了数据竞争和指针错误，确保程序在运行时不会访问无效内存地址。

生命周期是一种用于描述引用有效性和作用域的概念。生命周期连接了变量和引用之间的关系，确保引用在其指向的数据有效的情况下才能使用，并且在数据被释放后不会访问无效内存。Rust 的编译器可以在编译时进行生命周期检查，从而确保引用的有效性和内存安全性。

通过结合所有权系统、借用检查器和生命周期管理，Rust 实现了内存安全保障，使开发者可以编写高性能且不易出现内存错误的程序。这种设计使得 Rust 成为一种非常强大和安全的系统编程语言，在开发复杂、高性能应用程序时具有显著的优势。

2. 零成本抽象

Rust 语言的一个重要特性是“零成本抽象”（zero-cost abstractions），这种特性使得在 Rust 中使用高级抽象特性时不会损失性能。这意味着即使使用了以高级特性编写的代码，底层的执行效率也不会受到影响。

在 Rust 中，零成本抽象特性的实现方式主要体现在以下几个方面：

（1）零成本抽象的通用实现：Rust 通过编译器优化和代码生成，将高级抽象特性转换为底层机器码，以保证最终程序的性能。这意味着开发者可以使用泛型、模式匹配、trait 等高级特性，而不必担心性能损失。

（2）泛型编程：Rust 支持泛型编程，开发者可以编写适用于多种数据类型的通用代码。在 Rust 中，泛型代码在编译时会生成专门化的代码，针对具体的数据类型进行优化，而不会产生额外的运行时开销。

（3）模式匹配：Rust 中的模式匹配机制是一种强大的抽象工具，能够对数据结构进行解构和匹配。模式匹配在编译时会被转换为高效的机器码，不会引入运行时开销。

（4）trait 和实现：Rust 中的 trait 为抽象类型提供了统一的接口，同时在编译时将具体实现的 trait 转换为对应的代码。这种机制确保了代码的高效性和安全性，在运行时不会引入额外的开销。

Rust 通过采用严格的类型系统和编译器优化技术，实现了零成本抽象特性。这意味着开发者可以使用高级抽象特性来编写代码，而不必担心性能损失。这种特性使得 Rust 在保证代码高效性的同时，提供了丰富的抽象工具，使开发者能够更加灵活和高效地编写复杂的程序。

3. 并发性

Rust 的并发支持在其本身所提倡的“零成本抽象”的理念下，兼顾了效率和安全性。Rust 提供了一种被称为“信任三角”（The Three Pillars of Rust）的模型，其中包括内存安全、数据竞争安全和线程安全。Rust 的并发性主要通过以下几个方面来体现：

（1）线程安全：Rust 的所有权系统和借用规则确保了并发环境中的线程安全。Rust 的并发模型基于“消息传递”，这意味着线程之间通过消息传递进行通信，而不是共享可变状态。这样可以避免数据竞争和锁的使用，提高并发程序的安全性和可靠性。

（2）并发原语：Rust 提供了并发编程所需的原语，如 std::sync::mpsc（多生产者多消费者模型）、std::sync::Mutex 和 std::sync::RwLock 等，帮助开发者在并发环境中进行线程间通信和数据共享。

（3）并行性：Rust 的标准库中提供了 std::thread 模块，允许开发者创建线程并进行并行执行。此外，Rust 还支持操作系统线程直接与轻量级线程（std::thread::spawn）并发执行。

（4）并发设计模式：Rust 支持传统的并发设计模式，如 Future、Actor 模型，以及并发数

据结构等。并发设计模式可以帮助开发者在并发环境中更好地组织和管理代码。

（5）原子操作：Rust 提供了原子操作和内存顺序设置，通过 std::sync::atomic 模块，确保多线程之间对共享数据的操作是按照一定顺序进行的，避免出现数据竞争的情况。

通过以上方式，Rust 的并发支持帮助开发者编写出高效、安全的并发程序。Rust 的并发性设计基于内存安全和数据竞争检查的特性，使其同时具备高性能和并发安全性。

Rust 作为一种系统级编程语言，具有独特的内存安全、数据竞争安全和线程安全特性，通过“信任三角”模型确保并发程序的安全与可靠。其严格的所有权系统和借用规则保证了线程安全性，同时丰富的并发原语和设计模式为开发者提供了灵活的并发编程工具。通过并行性支持和原子操作，Rust 提供了一种高效、安全的并发编程方案。综上所述，Rust 的并发设计基于其独特的安全特性，使其成为一个非常具有前景的系统级编程语言，同时也为开发者提供了构建并发应用程序的良好基础。

1.3　Rust 语言发展历程和现状

Rust 语言的发展历程可以总结为以下几个阶段：

初始阶段（2010—2015 年）：Rust 语言在发布初期受到了一些关注，但没有引起太大的轰动。这个阶段主要集中在开发和完善语言的基本功能和工具链。

快速发展阶段（2016—2019 年）：随着 Rust 语言的稳定性和成熟度不断提高，越来越多的开发者和组织开始关注并使用 Rust。一些知名的项目，如 Firefox 浏览器和 Dropbox 等，也开始在部分代码中使用 Rust。

持续壮大阶段（2020 年至今）：Rust 语言的社区规模在不断扩大，开发者和项目数量也在快速增长。Rust 成为越来越多公司和组织的首选，用于编写高性能、安全的系统级应用程序。

经过多年的开发和改进，Rust 成为一个备受关注的编程语言，具有许多优势，例如在内存安全方面表现优秀。通过引入所有权系统、借用检查和生命周期检查等机制，Rust 有效避免了内存管理方面的常见错误。此外，Rust 还支持并发编程，提供通道、原子操作等机制，使并发编程更加方便和安全。同时，Rust 的编译器能够对代码进行静态分析，发现潜在的 bug（漏洞）和性能问题，从而提供更高质量的代码。

目前，许多行业领头公司正在使用 Rust 编程语言：

- Mozilla：Mozilla 是 Rust 语言的主要支持者，Rust 最初由 Mozilla 公司开发。Mozilla 已经在多个项目中使用 Rust，包括 Firefox 浏览器和 Servo 引擎等。Rust 语言的安全性和性能使其成为 Mozilla 的首选。
- Dropbox：作为一家知名的云存储和文件同步服务提供商，Dropbox 在一些后端服务中使

用 Rust 语言，以提高系统的性能和安全性。Rust 的并发性和内存安全性使 Dropbox 能够编写更加可靠和高效的代码。

- Amazon Web Services（AWS）：AWS 是全球领先的云计算服务提供商，他们在一些关键服务中使用 Rust 编程语言。例如，AWS Firecracker 是一个基于 Rust 编写的轻量级虚拟化技术，用于提供安全、高性能的云计算服务。
- Facebook：作为世界上最大的社交平台之一，Facebook 在一些关键系统中也使用了 Rust 语言。例如，Facebook 的 Libra 区块链项目就是使用 Rust 语言开发的，以提高系统的安全性和性能。
- GitHub: 全球最大的开源代码托管平台。GitHub 也在一些关键系统中使用 Rust 编程语言。例如，GitHub 的推荐引擎和安全扫描工具等都是使用 Rust 语言开发的，以提高系统的稳定性和性能。

在嵌入式领域和芯片设计领域，许多公司也开始使用 Rust 编程语言：

- Arm：全球领先的半导体技术公司 Arm 在嵌入式系统和物联网领域非常活跃。Arm 近年来开始支持 Rust 语言，并在一些项目中使用 Rust，以提高其嵌入式系统的性能和安全性。
- NVIDIA：知名的图形处理器制造商 NVIDIA 在人工智能、高性能计算和嵌入式系统领域占据重要地位，NVIDIA 在一些项目中使用 Rust 语言进行开发，以利用其并发性和性能优势。
- Intel：作为全球最大的芯片制造商之一，Intel 在嵌入式系统和云计算领域拥有广泛的影响力。虽然 Intel 主要使用 C 和 C++等传统编程语言，但他们也在一些项目中开始尝试使用 Rust 语言，以提高系统的安全性和性能。
- Infineon：全球领先的半导体制造商 Infineon 在 2023 年 3 月 7 日宣布其 32 位微控制器 AURIX™系列、TRAVEO™ T2G 系列和 PSoC™ MCU 系列支持 Rust 语言，成为全球领先正式支持 Rust 的半导体公司。
- 华为：作为通信业界领先公司和 Rust 基金会的创始成员，华为致力于推动 Rust 在通信软件行业的发展，并将持续为 Rust 社区做出贡献。华为之前在通信系统软件的开发中主要使用 C/C++代码，而 Rust 库使得 C/C++到 Rust 的迁移更加顺畅。华为在中国深圳举办了第一届 Rust China Conf 大会，并组织多项社区活动，包括为中国的开发者提供 Rust 教程和编码规范。

不仅如此，Rust 还拥有一个活跃且快速发展的开源社区，推动 Rust 编程语言的发展和应用。自 2010 年首次发布以来，Rust 语言迅速吸引了大量开发者的关注和参与，如今已经拥有数十万活跃的开发者和数千个项目。

Rust 社区拥有丰富和完善的生态系统，包括 Cargo 包管理器、crates.io 包索引、Rustup 工具链等，这些资源为开发者带来了巨大的便利，使他们能够更快地开发、构建和分享项目。此外，Rust 社区非常注重合作与分享，开发者们经常在各种线上和线下活动中相互交流、学习

和合作。Rust 编程语言也以其友好和支持性的环境而闻名，使新手和经验丰富的开发者都能获得帮助和支持。

Rust 社区在不断推出新的特性、工具和框架的同时，也在不断完善和改进语言本身，以提高开发效率和代码质量。通过定期的版本发布，Rust 不断解决 bug、增加功能和改进性能，推动 Rust 语言不断演进，为开发者提供更好的编程体验。

综上所述，基于 Rust 语言的诸多优势和逐渐增长的用户群体，可以预见 Rust 在未来将继续快速发展，并在各个领域得到更广泛的应用。随着更多行业对 Rust 的接触和认可，Rust 有望成为未来主流的编程语言之一，从而推动软件开发领域的进步和创新。

1.4　Rust 语言与 C/C++的比较

Rust 语言在设计之初的目的是取代 C/C++。经过多年的发展，Rust 语言逐渐展现出强大的特点和优势，使其成为一个可以真正挑战 C/C++地位的编程语言。Rust 语言有以下几个优势。

1. 内存安全

Rust 在设计时着重考虑了内存安全性，通过借用检查器和所有权系统来防止内存泄漏、指针悬空和数据竞争等问题。相比之下，C/C++没有内建的内存安全性检查机制，程序员需要手动管理内存，容易出现潜在的内存错误。Rust 的设计使开发者可以更加放心地编写代码，从而减少因为内存错误而导致的程序崩溃或安全漏洞的风险。

2. 并发安全

Rust 具有内建的并发性支持，通过线程安全的编程模式和独特的所有权系统来保证并发程序的安全性。虽然 C/C++也支持并发编程，但通常需要使用底层的线程和锁等机制，容易出现并发相关的 bug。因此，Rust 的内建并发性支持不仅提高了程序的安全性和可靠性，也减少了开发过程中出现并发问题的概率，使编写并发程序变得更加舒适和高效。

3. 语法和语义

Rust 的语法更加现代化和简洁，可以减少代码量并提高可读性，适合当今的软件开发，能够帮助开发人员更高效地构建可靠的软件系统。相比之下，C/C++的语法较为烦琐，需要更多代码来实现相同的功能，因此在某些情况下可能不如 Rust 优越。

4. 性能

C/C++通常被认为是性能优越的语言，能够进行高效的编译和优化。Rust 也被设计为高性能语言，通过一些优化技术和零成本抽象能力，可以实现接近 C/C++的性能水平。

不仅如此，在嵌入式领域中，Rust 除了在内存安全性、并发性和零成本抽象等方面具有明显优势之外，相较于 C/C++，它还有其他一些明显优势：

- 可移植性：Rust 具有良好的跨平台支持，不仅支持多种架构和操作系统，还能编写可移植的嵌入式代码。无论是在 ARM、x86、MIPS 等不同架构下，还是在 Linux、Windows、macOS 等不同操作系统中，开发者都能轻松使用 Rust 进行跨平台开发。
- 此外，Rust 的语法和标准库设计也为跨平台移植提供了便利。其严格的类型系统和模块化设计使代码更加清晰和易于维护，而标准库中提供的丰富功能则减轻了开发者在不同平台间切换时的工作量。因此，无论是开发桌面应用程序、移动应用程序还是嵌入式系统，使用 Rust 进行开发都能极大地提高代码的可移植性和可维护性。
- 社区和工具支持：Rust 语言具有强大的开源社区，其中有许多优秀的项目和库，为开发者提供了丰富的资源和工具。无论是初学者还是有经验的开发者，都可以在这个活跃的社区中找到合适的位置，学习和分享经验。不仅如此，开源社区还提供了丰富的文档和教程，为 Rust 编程语言的持续改进和更新提供了重要支持。许多优秀的开发者也在不断提交代码、修复 bug，为 Rust 语言的发展做出了重要贡献。

总的来说，尽管 C/C++在嵌入式领域有着长期的应用历史和丰富的生态系统，但 Rust 在内存安全性、并发性和零成本抽象等方面具有明显优势，使其成为一个具有潜力的选择，尤其适合对系统安全性和可靠性要求较高的项目。随着 Rust 在嵌入式领域的发展，它有望成为嵌入式系统开发的重要工具之一。

此外，Rust 语言与 C 语言之间可以良好共存和互操作。Rust 和 C 代码之间的互用性主要依赖于数据的互相转换。在 stdlib 标准库中提供了两个专用模块来支持这种转换，分别是 std::ffi 和 std::os::raw。

- std::ffi 模块：提供了一系列工具用于转换复杂类型，例如将 Rust 的字符串（&str 和 String）映射为 C 语言更易处理的类型。
- std::os::raw 模块：处理基本的底层类型，如整数和字符类型。由于 Rust 和 C 语言在内存布局上具有足够的相似性或一致性，因此 Rust 编译器可以隐式地转换这些基本类型。

尽管 Rust 在网络应用程序和裸机嵌入式系统中表现出色，但其生态和函数库资源相对有限。不过，与 C 之间的互通性使 Rust 更容易集成到现有软件中，从而加速了这类软件的部署和应用。

虽然 Rust 具有内存安全性，但在与 C 代码结合时可能会存在安全隐患。因此，为确保 Rust 的安全性，有必要在 C 和 Rust 之间找到合理的分界点。

总的来说，Rust 在嵌入式系统中替代 C 和 C++的适用性已经得到证实，特别是在新代码开发方面。除了内存安全外，Rust 强大的类型系统和错误处理也为其带来了卓越的可读性和可维护性，同时在不增加额外开发成本的情况下实现了更安全的代码。

1.5　Rust 语言学习资源

本章对 Rust 语言进行了简单介绍，内容包括 Rust 语言的特性、发展历程和现状、与 C/C++ 的对比等。希望读者可以通过本章快速了解 Rust 语言的基本特性和优势。然而，由于 Rust 语言的复杂性和深度，本章仅为入门级别的介绍。如果读者需要深入且系统地学习 Rust 语言，可以通过以下官方网站和社区资源进行学习。

（1）Rust 官方文档（https://doc.rust-lang.org/）：

Rust 官方文档是学习 Rust 编程语言最全面和权威的资源，涵盖语言特性、标准库、工具等内容，对于入门者和进阶者都非常有帮助。

（2）图书 *The Rust Programming Language* (https://doc.rust-lang.org/book/)：

这本书由经验丰富的 Rust 程序员撰写，帮助学习者系统地掌握 Rust 的各种特性和用法。

（3）Rust Reddit 社区（https://www.reddit.com/r/rust/）：

Reddit 上有一个专门讨论 Rust 编程语言的社区，用户可以在这里向其他 Rust 程序员请教问题，分享经验，了解最新的开发动态等。

（4）Rust 编程语言官方论坛（https://users.rust-lang.org/）：

Rust 编程语言官方论坛是一个专门讨论 Rust 相关话题的平台，用户可以在这里找到很多有用的信息和资源，与其他 Rust 程序员交流、互动。

（5）Rust 编程语言 GitHub 仓库（https://github.com/rust-lang/rust）：

Rust 编程语言的开源项目托管在 GitHub 上，学习者可以通过查看源码、提出问题、提交代码等方式与 Rust 社区的朋友互动，加深对 Rust 的理解和掌握。

在接下来的章节中，我们将逐步介绍如何使用 Rust 语言进行针对汽车电子的嵌入式应用开发。首先，从嵌入式系统的基本概念开始讲解，包括嵌入式系统的特点、架构、硬件接口等内容，帮助读者快速了解嵌入式开发的基本知识。然后，我们将介绍如何在嵌入式设备上运行 Rust 程序，包括配置开发环境、交叉编译 Rust 程序等内容，以便读者可以顺利地在嵌入式平台上进行开发和调试。

在此基础上，我们将介绍如何利用 Rust 语言实现嵌入式系统的应用以及汽车电子实战开发。结合具体的实例和案例，帮助读者更好地理解和应用 Rust 语言在嵌入式领域的优势和特点。通过实际的项目开发，读者将能够深入学习和掌握使用 Rust 语言开发高效、安全的嵌入式应用程序的方法和技巧。

1.6　总结与讨论

大周和小张通过系统地学习 Rust 语言，已经对 Rust 语言有了一定的了解。于是，他们找到 Tom 进行总结汇报。在听取了他们的汇报后，Tom 对“Rust 是否适合在汽车电子行业替代

C/C++”这一问题有了初步的了解。他们一起讨论并发表了各自的想法，最后对 Rust 语言能否解决当前困境做了总结。

Tom: “Rust 作为一个较新的语言，必然有后发优势。与传统的 C/C++相比，它确实拥有一些独特的优势。Rust 提供了内存安全的保证，这是它在嵌入式开发领域中引人注目的一大亮点。通过所有权和生命周期的概念，Rust 能够有效防止内存泄漏和指针错误。此外，Rust 的并发模型设计得很好，有助于构建稳定的多线程应用，这在汽车电子软件开发中非常重要。由于 Rust 语言拥有异常强大的编译器和语言特性，其代码天然就比其他语言拥有更少的 bug。同时，Rust 拥有非常完善的工具链和良好的包管理工具，使其非常适合汽车电子软件这种大型团队的协作开发。”

大周: “Rust 采用的编译时错误检查系统能够有效捕捉到许多在 C/C++中可能要等到运行时才能暴露的问题，这是其一大优势。此外，Rust 实现了零成本抽象的概念，这意味着我们可以在编写高级语言特性的同时保持高效的性能。这种优势使得 Rust 成为一个理想的选择，既满足了对安全性的需求，又不牺牲性能。不仅如此，Rust 的包管理工具 Cargo 也为开发人员带来了极大的便利。Cargo 不仅可以管理项目的依赖关系，还可以自动构建项目并管理发布的版本，极大地简化了软件开发过程。总的来说，Rust 在汽车电子软件开发中能够带来很大的便利，使用 Rust 是一个理想的选择。”

小张: “Rust 能够让我深入理解内存、堆栈、引用、变量作用域等其他高级语言往往不会深入接触的内容。在编写 Rust 代码时，语法、编译器和 Clippy 这些静态检查工具会让我写出更好的代码。Rust 的学习和实践过程让我对程序设计和开发有了更深入的理解，也提高了我的编程技能和思维能力。通过更好地利用 Rust 这个强大的编程语言，我能够提升自己在汽车电子软件开发领域的能力和水平。”

1.7 练习

1. 检查示例代码中的问题，并发现其中的内存泄漏问题。
2. 找到自己手头已有的嵌入式开发板，列出其 CPU 类型、内存大小和存储大小。
3. 列出 C/C++语言常用的内存申请、内存释放、复制等指令，并理解其参数的含义。